BLOCKMAKER LIVE

Guía de ejercicios

Módulo 7

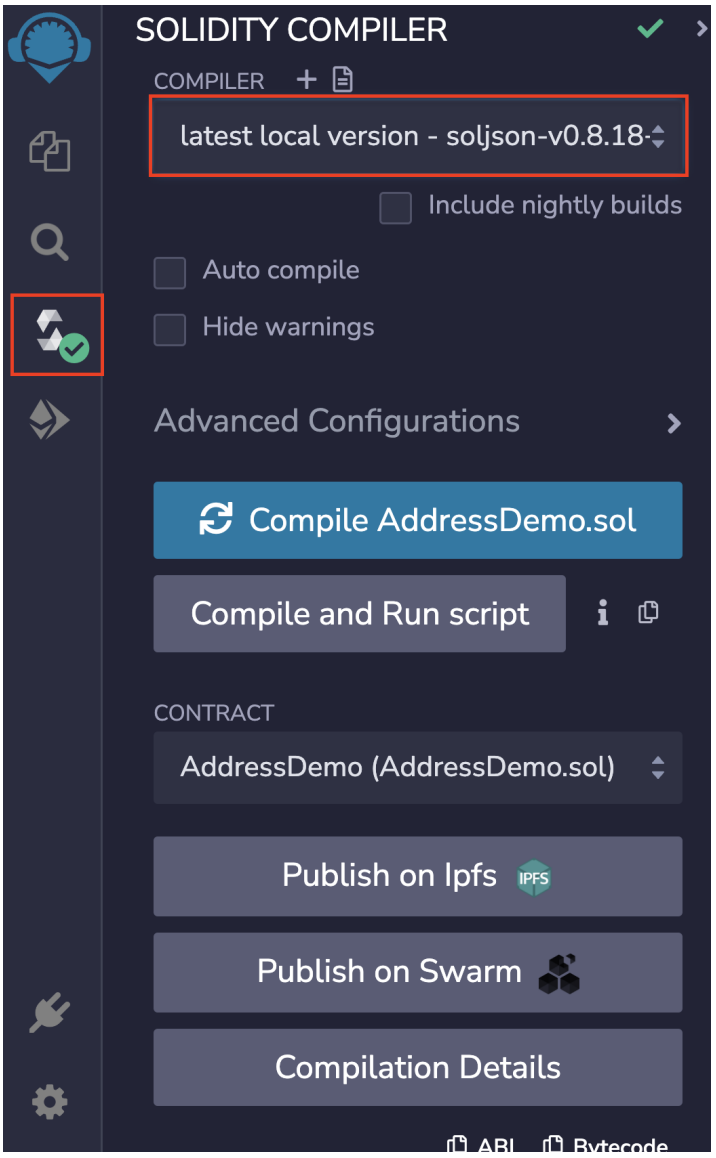
Recordatorios

1.A continuación, os dejamos un enlace a modo de bibliografía y de utilidad para consultar cualquier duda en materia de Solidity (si lo necesitáis, usar la función de traducir página al español):

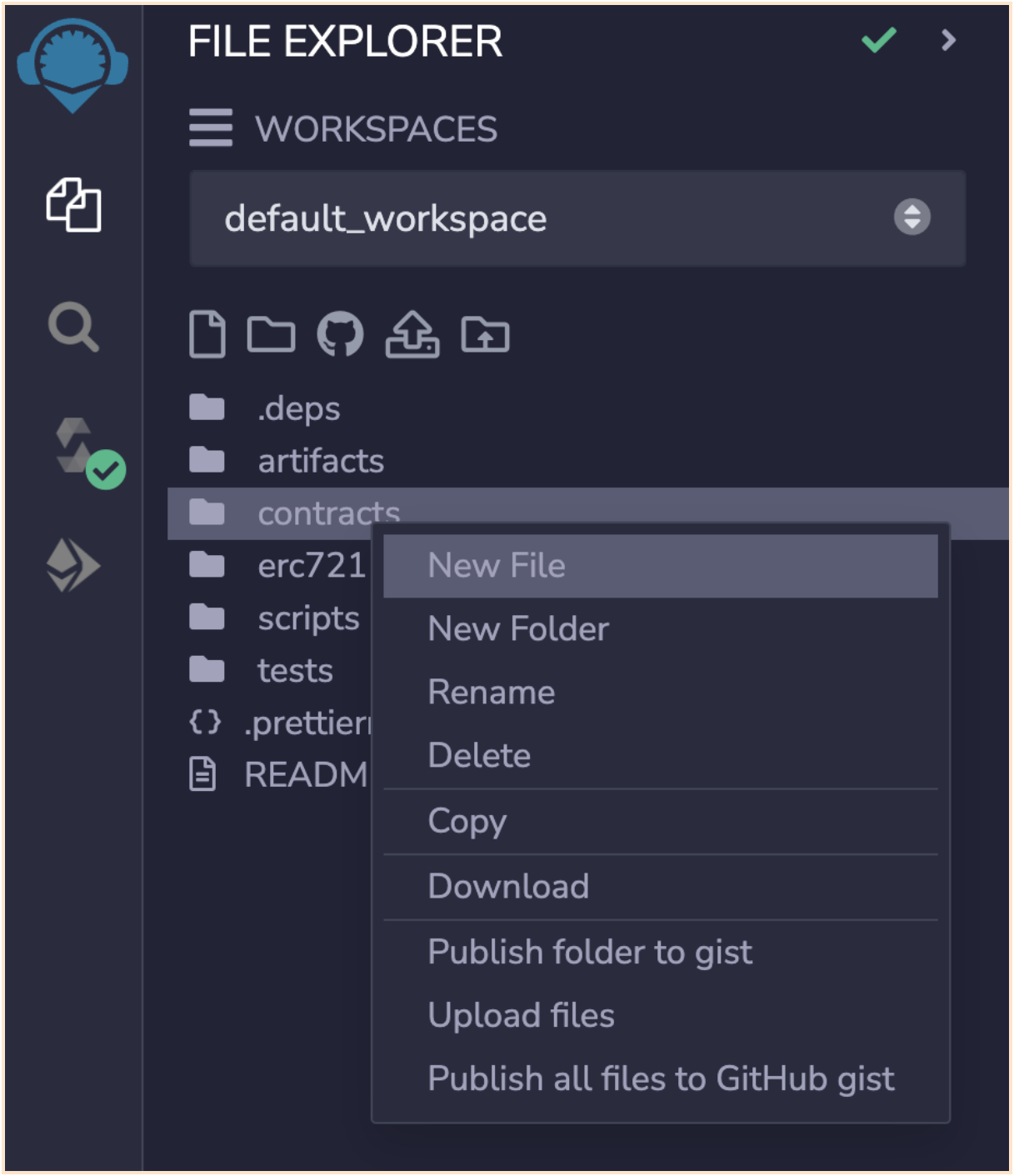
- https://docs.soliditylang.org/en/v0.8.21/grammar.html

2.Para realizar estos ejercicios vamos a utilizar el compilador online de Remix http://remix.ethereum.org/

RECUERDA 1: Desde la página http://remix.ethereum.org/ vamos a irnos a la tercera sección de la izquierda y vamos a cambiar la versión del compilador. Para ello, seleccionaremos cualquiera superior a la 0.8.0:



RECUERDA 2: Para crear un nuevo contrato, nos iremos a la primera sección de la izquierda y colocaremos en la carpeta “contracts” -> “New File” y le pondremos una nombre con la extensión “.sol.



Paso 0. Crear el contrato y descripción general

Crea un nuevo contrato Solidity con el nombre "**Votacion**."

Descripción del contrato: El siguiente contrato muestra muchas de las características de Solidity. Implementa un contrato de votación. Por supuesto, los principales problemas de la votación electrónica son cómo asignar los derechos de voto a las personas correctas y cómo evitar la manipulación. No resolveremos todos los problemas aquí, pero al menos mostraremos cómo se puede delegar el voto para que el recuento de votos sea automático y completamente transparente al mismo tiempo.

- La idea es crear un contrato inteligente por papeleta, proporcionando un nombre corto para cada opción. A continuación, el creador del contrato, que actúa como presidente, otorgará el derecho de voto a cada dirección individualmente.
- Las personas detrás de las addresses pueden elegir votar ellas mismas o delegar su voto en una persona de su confianza.
- Al final del tiempo de votación, propuestaGanadora() devolverá la propuesta con el mayor número de votos.

Paso 1. Introducir cabecera contracto

Copiar al inicio del contrato inteligente.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

Paso 2. Declarar “contract” Votación

Creamos el **contract Votacion** (recuerda que se declara como si fuera una clase y se le añaden las llaves para introducir dentro todo el contenido).

Paso 3. Crear variable presidente

Define una **variable de estado llamada "presidente"** de tipo **address** y haz que sea **pública (public)**. Esta variable almacenará la dirección del propietario del contrato.

Paso 4. Crear variable Votante

Define una **variable de estado llamada "Votante"** de tipo **struct**.
Esta variable contendrá las siguientes propiedades:

- peso (entero): peso acumulado por el votante mediante delegación
- votado (boolean): si es true, la persona ya habrá votado
- delegado (address): persona delegada
- voto (entero): índice votado del futuro array de propuestas de la votación

Paso 5. Crear variable Propuesta

Define una **variable de estado llamada "Propuesta"** de tipo **struct**.
Esta variable contendrá las siguientes propiedades:

- nombre (string): nombre corto de la propuesta de votación
- cantidadVotos (entero): número de votos acumulado

Paso 6. Crear mapping votantes

Crea un mapeo llamado "**votantes**" que mapea **el address del votante** (**address**) con su **struct Votante** (**struct**) correspondiente. Este mapeo debe ser **público (público)**.
De esta menera, almacenamos la relación entre el propio votante y sus decisiones a nivel de voto.

Paso 7. Crear array de Propuestas

Crea un **array de Propuestas (Propuestas[])** llamado "**propuestas**" con visibilidad **pública** que almacenará todas las propuestas existentes de manera inicial (e.g., pongamos de ejemplo que son los candidatos a la presidencia).

Paso 8. Crear evento DerechoVotoOtorgado

Crea un **evento (event)** llamado "**DerechoVotoOtorgado**" con los siguientes datos:

- votante indexed (address)

Evento para registrar cuando se otorga el derecho de voto.

Paso 9. Crear evento VotoDelegado

Crea un **evento (event)** llamado "**VotoDelegado**" con los siguientes datos:

- remitente indexed (address)
- delegado indexed (address)

Evento para registrar cuando se delega un voto.

Paso 10. Crear evento VotoEmitido

Crea un **evento (event)** llamado "**VotoEmitido**" con los siguientes datos:

- votante indexed (address)
- propuestas indexed (entero)

Evento para registrar cuando se emite un voto.

Paso 11. Crear modificador soloPresidente

Crea un **modificador (modifier)** llamado "**soloPresidente**" el cual no requiere ningún tipo de variable de entrada y comprobará que el **emisor de la petición (msg.sender) es el presidente**, en caso contrario, fallará.

Paso 12. Crear constructor

Crea una **función constructora (constructor)** que contenga:

***Parámetros de entrada**:*

- Variable **nombresPropuestas** de tipo **string[]**. A través de la misma le pasaremos los nombre de todas las propuestas candidatas (en nuestro caso todos los candidatos a la presidencia).

***Descripción**:*

- Igualamos la variable **presidente** al propio emisor del despliegue**.**
- Ahora, deberemos introducir **todos los nombres de propuestas** que nos han llegado en el array de string (string[]) llamado **propuestas**.
    - Realizamos un **bucle for()** que **comience por 0** y **finalice cuando el valor de la variable que declaremos "i" sea >= nombresPropuestas.length** (longitud máxima del array).
        - Dentro del propio bucle, lo que haremos será introducir dentro del array **propuestas** un objeto/struct por nombre que nos ha llegado en **nombrePropuestas**. Para ello utilizaremos la función **push()** de los arrays, con el fin de introducirle las Propuestas (propuestas.push(<INTRODUCIR_STRUCT_PROPUESTA*).

Al introducir el objeto Propuesta, estableceremos el "nombre" como "nombrePropuestas[i]" (con la variable que recorremos el array de nombres) y la "cantidadVotos" la pondremos a 0.

Paso 13. Implementar función darDerechoAVotar

Implementar la función "**darDerechoAVotar**" **pública** que permitirá al presidente dar derecho a votar a addresses específicas.

***Parámetros de entrada**:*

- Variable **votante** de tipo **address**.

***Modificador de función**:*

- soloPresidente.

***Descripción**:*

- Comprobamos que el votante no haya votado, es decir, miramos dentro del mapping **votantes**, si el votante tiene la variable **votado** a "**true**".
- Comprobamos que al votante no se le haya asignado el derecho a votar previamente, es decir, miramos dentro del mapping **votantes**, si el votante tiene la variable **peso** a 0.
- Ahora añadiros al array de **votantes** al propio **votante** accediendo a su variable **peso** y asignándole valor **1** (votantes[votante].peso).
- Por último, emitimos el evento **DerechoDeVotoOtorgado**.

A pesar de que siempre hemos inicializado los struct con todos los datos que contienen, en Solidity no es necesario inicializar todas sus variables.

Paso 14. Implementar función delegar

Implementar la función “**delegar**” **pública** que permitirá delegar el voto a otra persona**.**

Parámetros de entrada*:*

- Variable **receptor** de tipo **address**.

Descripción*:*

- Comprobamos que el **emisor** de la petición tenga un **peso != 0** porque sino no podría votar ni delegar el voto (para ello accedemos al mapping de votantes).
- Comprobamos que el **emisor** de la petición tenga un tenga la variable **votado sea = false** porque sino ya habría votado (para ello accedemos al mapping de votantes).
- Comprobamos que el address del **emisor** no coincida con el address enviado como parámetro de entrada **receptor**.
- Realizamos un bucle **while** que compruebe si el receptor ha establecido ya un delegado de voto, mapping de votantes[receptor], es decir, si ese valor es distinto de 0.
    - Dentro del bucle estableceremos la variable **receptor = al delegado que tenga establecido el votantes[receptor]**, es decir, que si el receptor ya tiene un delegado elegido, ese será el nuevo receptor.
    - Finalmente, se realiza un comprobación para ver si el **receptor es distinto del emisor** de la recepción, ya que sino habría un bucle en la delegación
- Al salir del bucle, vamos a comprobar que el peso de **votantes[receptor] sea >= 1**, ya que esto nos dejará claro si tiene derecho a voto o no.
- Tras esto, vamos a establecer las variables **votado=true y delegado=receptor** de **votantes[msg.sender]** (del emisor que quiere delegar), ya que de este modo ya se habrá delegado el voto y aparece como voto completado.
- Finalmente, comprobamos en base a if/else:
    - If votantes[receptor] ya ha votado
        - Si el delegado ya votó, añade directamente al número de votos el peso del que delegó el voto (usar propuestas[votantes[receptor].voto] para acceder a la cantidadVotos e incrementarla en base al peso del receptor)
    - Sino, se incrementa el peso del delegado añadiéndole el peso del remitente. A votantes[receptor].peso le añadimos el peso de votantes[msg.sender].peso.
- Para finalizar, emitimos el evento de VotoDelegado con los parámetros correspondientes.

Paso 15. Implementar función votar

Implementar la función "**votar**" **pública** que permitirá votar a aquellos que puedan votantes a los que se les haya otorgado previamente el voto.

***Parámetros de entrada**:*

- Variable **proposición** de tipo **entero que representa el índice de la proposición a votar**.

***Descripción**:*

- Comprobamos que al votante se le haya asignado el derecho a votar previamente, es decir, miramos dentro del mapping **votantes**, si el votante tiene la variable **peso != 0 o >0**.
- Comprobamos que el votante no haya votado (del msg.sender), es decir, miramos dentro del mapping **votantes** si el votante tiene la variable **votado** a "**true**".
- Tras estas comprobaciones, ponemos al votante con la variable **votado** a **true**.
- Ponemos al votante con la variable **voto** = a la **proposición** realizada.
- Actualizamos la **cantidadVotos** dentro del array de **propuestas** el correspondiente al índice de la **proposición** (propuestas[proposicion]) que nos han pasado con el peso total que tenía el **votante** (votantes[msg.sender]).
- Por último, emitimos el evento **VotoEmitido**.

Paso 16. Implementar función propuestaGanadora

Implementar la función "**propuestaGanadora**" **private** que permitirá obtener el índice de la propuesta ganadora.

***Parámetros de salida**:*

- Variable de tipo **entero correspondiente al índice de la propuesta ganadora**.

***Descripción**:*

- Declaramos e inicializamos dos variables a 0 que serán la **cantidadVotosGanadores** e **indicePropuestaGanadora**.
- Recorremos con un bucle **for** el tamaño de las propuesta (propuestas.length).
    - Dentro de este, comprobaremos con un **if** si la **cantidadVotos de la propuesta** que estamos recorriendo (e.g., propuestas[p]) > **cantidadVotosGanadores**.
        - Si entra en dicho if, actualizamos **cantidadVotosGanadores** con **cantidadVotos de la propuesta que va ganando** y el **indicePropuestaGanadora** con el **índice de de la propuesta** que estamos recorriendo (e.g., "p").
- Fuera del bucle for, devolveremos el **indicePropuestaGanadora**.

Paso 17. Implementar función nombreGanador

Implementar la función "**propuestaGanadora**" **public** que devolverá el nombre de la propuesta ganadora.

***Parámetros de salida**:*

- Variable nombre de la **propuesta ganadora** de tipo **string**.

***Descripción**:*

- Esta función será tan sencilla como devolver el **nombre** de la **propuesta ganadora invocando a la función propuestaGanadora()** para obtener el índice de la misma (pista: **propuestas[propuestaGanadora()]**).